The CLARAty Architecture for Robotic Autonomy†

Richard Volpe
Issa Nesnas
Tara Estlin
Darren Mutz
Richard Petras
Hari Das

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91109
Phone: 818-354-4321
Email: *firstname.lastname*@jpl.nasa.gov

Abstract— This paper presents an overview of a newly developed Coupled Layer Architecture for Robotic Autonomy (CLARAty), which is designed for improving the modularity of system software while more tightly coupling the interaction of autonomy and controls. First, we frame the problem by briefly reviewing previous work in the field and describing the impediments and constraints that been encountered. Then we describe why a fresh approach of the topic is warranted, and introduce our new two-tiered design as an evolutionary modification of the conventional three level robotics architecture. The new design features a tight coupling of the planner and executive in one Decision Layer, which interacts with a separate Functional Layer at all levels of system granularity. The Functional Layer is an object oriented software hierarchy that provides basic capabilities of system operation, resource prediction, state estimation, and status reporting. The Decision Layer utilizes these capabilities of the Functional Layer to achieve goals by expanding, ordering, initiating and terminating activities. Both declarative and procedural planning methods are used in this process. Current efforts are targeted at implementing an initial version of this architecture on our research Mars rover platforms, Rocky 7 and 8. In addition, we are working with the NASA robotics and autonomy communities to expand the scope and participation in this architecture, moving toward a flight implementation in the 2007 time-frame.

TABLE OF CONTENTS

1. BACKGROUND OF THIS EFFORT

History Outside of JPL

The development of Robotics and Autonomy architecture is as old as the field itself. Therefore, it is not possible here to completely review the body of work upon which this effort builds. Instead, we will simply describe some of the more recent or dominant and trends influencing the new architecture presented in this document.

Efforts in robotic architectures have largely arisen from a pragmatic need to structure the software development for ease of system building. As such, they have grown in scope and complexity as the corresponding systems have grown. Early efforts concentrated in detailed software packages [19], or general frameworks [2]. Only in the last decade, with the emergence of fast computers with real-time operating systems, have infrastructures been designed as open-architecture controllers of modern robot systems [35][27][10].

In parallel with robot control efforts, artificial intelligence systems for planning/scheduling and execution were developed which relied on underlying closed-architecture robot controllers [15][29]. The tendency of these systems to be slow and computationally costly led to the emergence of a minimalist school of thought using Behavior Control [11]. But with faster control layers available, and a general desire to leverage planning functionality, newer systems implement a multi-tiered approach that includes planning, execution, and control in one modern software framework [1][3].

While these end-to-end architectures have been prototyped, some problems have emerged. First, there is no generally accepted standard, preventing leverage of the entire community's effort. This problem has lead to the second, which is that implemented systems have typically emerge as patchwork of legacy and other code not designed to work together. Third, robotics implementations have been slow to leverage the larger industry standards for object-oriented software development, within the Universal Modeling Language (UML) framework. Therefore, we believe the time is ripe to revisit robotics and autonomy efforts with fresh effort aimed at addressing these shortcomings.

History Inside of JPL

The Jet Propulsion Laboratory, California Institute of Technology (JPL) has a long history in building remotely commanded and controlled machines for planetary exploration.

† 0-7803-6599-2/01/$10.00 © 2001 IEEE

Most of this effort has has concentrated on very simple and robust execution of linear sequences tediously created by ground controllers. Areas where expertise has concentrated on sophisticated on-board closed loop control, have been largely outside of the traditional areas of robotics, falling instead in the realm of aerospace guidance and navigation. Further, the implementation of these solutions have been in hand tailored software solutions, optimized for specific spacecraft and limited CPU and memory. Only more recently have concepts from robotics and autonomy started to be used or considered for flight missions [24][23].

Therefore, the history of robotic efforts at JPL has been primarily within the research program. The oldest of these efforts were in the areas of manipulator and teleoperation systems, and had limited software or software architecture components [8]. One of the first major software architecture efforts was within the Telerobotic Testbed, a large research effort for developing autonomous, multi-robot, satellite servicing [7]. While a very complex system conforming to the NASREM architecture [2], it relied on several subsystems using disparate software paradigms. Except through the diffuse efforts of the individual research participants and their subsequent assignments, little of this software structure survive the demise of the Testbed. Afterward, many smaller on-orbit manipulator research projects existed, each with their own software implementation: Remote Surface Inspection (C and VxWorks), Satellite Servicing (C and and assembly), MOTES (Ada and VxWorks), etc. [34][9][5]. Each of these efforts provided parallel duplication of similar functionality with minimal code sharing due to architectural differences.

In parallel with these robot manipulation efforts were several mobile robot efforts, each developing software infrastructure in relative isolation. At about the same time as the Testbed, there was the development of a large Mars Rover platform name Robby, using C and VxWorks [37]. Research with Robby ended as there was a paradigm shift from large rovers with software for deliberative sensing and planning, to small rovers with reactive behaviors [18]. The fourth of these *"Rocky"* vehicles, programmed with C and Forth and without an underlying operating system, sold the concept of placing the Sojourner rover on the Pathfinder Mission. However, Sojourner itself, was programmed with software written from scratch, not inherited from its predecessors.

Only as Sojourner was being built, did a new rover research begin to address the problem of providing a software infrastructure with modularity, reconfigurability, and code re-use implicit in the design. To this end, a new rover, *Rocky 7*, was built, and its development team selected the ControlShell C++ software development environment [27][33]. Unfortunately, as subsequent research rover efforts were started, a new spectrum of control infrastructures re-emerged in rover tasks (e.g. FIDO, DARPA TMR, Nanorover, etc), repeating the duplication of efforts seen in manipulation tasks half a decade before [26][40][32].

In the same time frame as the construction of Sojourner and *Rocky 7* was a large scale effort in Autonomy and Control for flight, but targeted for cruise and orbit, not surface operations. Under the aegis of the Deep Space One project and later renamed the Remote Agent Experiment (RAX) [24], this was a collaborative effort between JPL and NASA Ames Research Center (ARC). Emerging from it, was a determination at JPL to build a fundamentally new software architecture for all future missions, name the Mission Data System [13]. MDS is an object-oriented, state based architecture, and moves radically away from all previous mission control concepts which are sequence based. While it was originally targeted for orbital insertion and outer-planet mission, it is now addressing a Mars surface mission scheme for its first use.

Therefore, given the large efforts in software architecture development at JPL under the MDS flag, and given the history of divided efforts in the robotics research community, it is the objective of authors of this report to put forth a new framework for robot software at JPL and beyond. This report outlines the results for the first year, describing the broad design of the resultant CLARAty architecture, providing some initial implementation efforts, and outlining the directions for upcoming construction of end-to-end rover control software under this new framework.

2. The Challenge

Having briefly reviewed the history of robot control architectures, it is apparent that more work is required. In this section we will summarize the impediments to success that have existed in the past, outline the reasons for attempting to overcome them with a new architecture, and describe the constraints on the solution to be provided.

Impediments to Success

There are numerous impediments to the success of control frameworks for robotics systems. These may be categorized as follows:

Programmatic Vision — Implicit in the success of any research endeavor is the need to sustain the effort with funding, especially early in its development. Typically it has been difficult to maintain significant research funding for control architecture development. This is primarily because the end product is infrastructure, not a new robot system or algorithm. While this new infrastructure might enable better or faster system and algorithm development, such indirect results have been difficult to sell programmatically.

Not Invented Here (NIH) — For the most part, autonomy and robotic systems are still in the domain of research products, and not commercial products. Therefore, it is typical for each research team to to want to develop and grow its own products. This expresses their inventiveness, as well as giving their work a unique signature used in promotion of their results.

Fear of unknown products — Closely tied to NIH, is the fact that research products from outside of one's team have varying and unknown levels performance, quality, and support. Therefore, use of other's products might not only dilute ones research identity, but consume valuable effort while trying to adopt it.

Flexibility — Also, because of the research nature of robotics, there is still no absolute consensus on how to best solve the problems that exist, or even which are the most important problems to solve. Therefore, researchers often desire maximum flexibility from their hardware and software, to meet the specific needs of new projects. This desire for flexibility is often at odds with any software framework that is not specifically tailored to the task. Simply put, no one architecture can be optimal for all problems, and if one is too flexible it quickly loses any structure that gives it value.

Overhead — Often coupled to a desire for flexibility is a need to optimize performance. This comes in the form of computational overhead for the robot system, as well as system building overhead, encountered in the use of software development products which are unfamiliar or unwieldy.

Critical Mass — Even if a new software infrastructure is recognized as being valuable, that value might not be realized unless a large enough group of researchers chooses to standardize around it. Once such a group exists and provides critical mass, the standard enables much easier exchange of software and ideas, which in theory can "snowball". However, it is a difficult decision for any one research team to join a new standard until critical mass has been reached. This is because any external standard will require overhead, while the benefits may only come after critical mass is achieved.

Learning Curve — Human nature and conservative logistics of any research program provide a resistance to abandoning well known and understood methods for new ones that require an investment of time to learn. This is especially true when projects are on short development cycles, which has been more true in recent years.

Technical Vision — Because most researchers have had to develop infrastructure to build their systems, they have developed opinions about their preferred solutions. While some are willing to abandon these solutions in favor of an external product, others have a technical vision which may be at odds with external products, no matter how mature. Depending on the strengths of their convictions, some researchers may not join the larger community in the use of a standard architecture (unless it is their own). While this may ultimately be detrimental to their research, it is also detrimental to the community as well.

Needs for a New Start

Given these impediments to the acceptance of a unifying architecture, one may wonder why there should be an impetus for its creation. The primary reason is that which drives the desire for robotics in the first place: elimination of the need for people to waster their time on lesser endeavors. There are three paths to this goal.

1. Elimination of duplicative efforts which prevent attainment of critical mass:

Parallel Duplication — As previously discussed, there are often duplicative efforts within both robotic manipulation and mobility research. This diminishes the final products by wasting resources on solving the same problems, in different ways, at the same time.

Serial Duplication — It is also evident that as new research tasks start, they often wipe the slate clean to eliminate old system problems and lack of familiarity or trust with previous products. Typically, the only software with legacy is due solely to a single individual, not the local or extended community. Obviously, without the ability to bridge to group ownership, transfer outside of individual institutions is even more restricted.

2. Follow software community lead:

Open source movement — The value of shared software has been dramatically illustrated by Linux, GNU, and other share/free ware products. Typically this has existed within the desktop PC market, but there is no obvious reason why this model cannot be leveraged by research software within the robotics community. As evidence of this fact, there has recently been an announcement for Intel sponsorship of an open source Computer Vision Library [20].

Object oriented design — Complementary to the open source movement, has been the growth of object oriented design for PC software. In much of the commercial software industry it dominates. However, this paradigm is largely under-utilized in robotics, isolating the community.

3. Leverage complimentary efforts:

Software sharing — To build critical mass amongst a worldwide but relatively small robotics community, it would be extremely beneficial to have an architecture framework that was widely accepted. Not only would this enable easier sharing of design concepts, but, more importantly, it would enable the direct transfer of software to all parties. Even sharing among the limited communities of JPL and NASA is currently arduous and therefore rare. A first step would be to eliminate these hurdles completely.

Mission Data System and X2000 — Recently, NASA has invested heavily in large scale efforts in spacecraft hardware (X2000) and software (MDS) which promise an infrastructure to be leveraged and expanded [38][13]. It is to the benefit of NASA robotics efforts to also use these products where applicable. Since the spacecraft control problem is very similar to the general robotics problem, it is anticipated that there is much to be gained by this leveraging. Obviously other sources of relevant technology will exist outside of this limited set, and will be incorporated when applicable.

Constraints on the Solution

Given these needs, there are several issues that will constrain the success of an architectural solution. First, there is a need for Community Acceptance. Without acceptance by the robotics and autonomy community, both from users and developers, there can not be a success. Full acceptance is probably not possible, or even desirable in a growing research area. However, as described previously, it is important to reach a level of critical mass, so that users and developers gain more than they lose from adherence to standards and participation in software exchange.

Second, it is vital to span the many divides within the necessary user and developer communities. These divides exist in many forms, between and within robotics and AI research areas. They can result from a desire to solve different types of robotics problems, all the way from parts assembly to humanoid interfaces. Or they can result from an emphasis on different phases of product life cycles, from basic research to fielded system. Within and across institutions, the differences can be cultural as well, spanning departments from mechanical engineering to computer science, and organizations from academia to commercial companies.

Third, there is a required need to leverage existing software in research and NASA flight efforts. In particular, at JPL there has been a substantial effort in the new MDS, which is very similar to the architecture work described herein but has been largely focused on the problems of zero gravity spacecraft, not robots operating on planetary surfaces.

Finally, it is a requirement to leverage standard practices in industry. This is needed to avoid reinvention of the wheel, and enable NASA robotics efforts to adopt techniques and solutions commonly employed in commercial products, and within the global software community.

3. THE CLARATY ARCHITECTURE

In response to these needs and requirements we have developed the initial framework for a new Autonomous Robot software architecture. This section will review this new structure, discuss its evolutionary differences from its predecessors, introduce each layer of the architecture, and provide an overview of the interaction between them.

Review of Three Level Architecture

Typical robot and autonomy architectures are comprised of three levels — Functional, Executive, and Planner as shown in Figure 1 [17][30][1].

The dimension along each level can be thought of as the breadth of the system in terms of hardware and capabilities. The dimension up from one layer to the next can be thought of as increasing intelligence, from reflexive, to procedural, to deliberative. However, the responsibilities and height of each level are not strictly defined, and it is more often than not the case that researchers in each domain expand the capabilities and dominance of the layer within which they are working. The result are systems where the Functional Layer is dominant [28][36], or the executive is dominant [30][10] or the the planner is dominant [15][14]. Further, there is still considerable research activity which blurs the line between Planner and Executive, and questions the hierarchical superiority of one over the other [21][16].

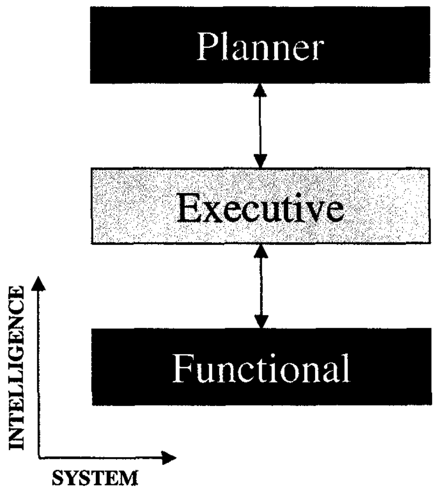

Figure 1. Typical three level architecture.

Another problem with this description is lack of access from the Planner to the Functional Level. While this is typically the desirable configuration during execution, it separates the planner from information on system functionality during planning. One consequence is that Planners often carry their own separate models of the system, which may not be directly derived from the Functional Level. This repetition of information storage often leads to inconsistencies between the two.

A third problem with this description is the apparent equivalence of the concepts of increasing intelligence with increasing granularity. In actuality, each part can have its own hierarchy with varying granularity. The Functional Layer is comprised of numerous nested subsystems, the executive has several trees of logic to coordinate them, and the planner has several time-lines and planning horizons with different resolution of planning. Therefore, granularity in the system may be misrepresented by this diagram. Worse, it obscures the hierarchy that can exist within each of these system levels.

Proposed Two Layer Architecture

To correct the shortfalls in the three level architecture, we propose an evolution to a two-tiered Coupled Layer Autonomous Robot Architecture (CLARAty), illustrated in Figure 2. This structure has two major advantages: explicit representation of the system layers' granularity as a third dimension [1], and blending of the declarative and procedural techniques for decision making.

The addition of a granularity dimension allows for explicit representation of the system hierarchies in the Functional Layer, while accounting for the *de facto* nature of planning horizons in Decision Layer. For the Functional Layer, an object oriented hierarchy describes the system's nested encapsulation of subsystems, and provides basic capabilities at each level of the nesting. For instance, a command to "move" could be directed at a motor, appendage, mobile robot, or

[1] The convention employed here is to consider lower granularity to mean smaller granual sizes.

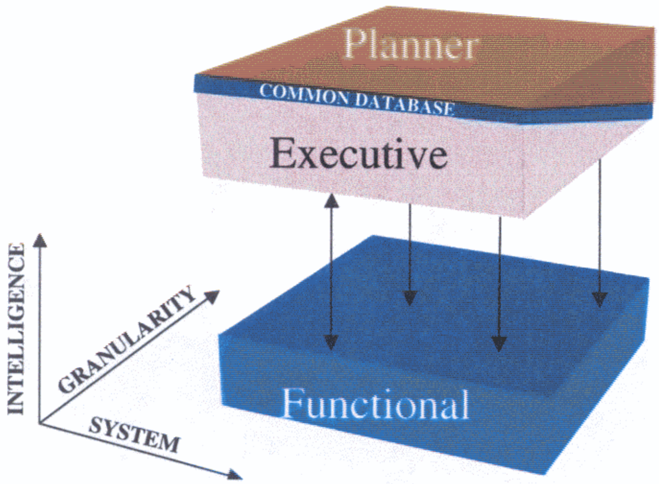

Figure 2. Proposed two layer architecture.

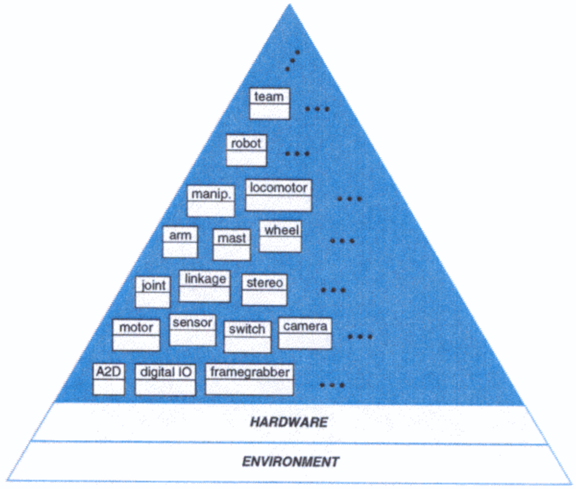

Figure 3. Proposed Functional Layer.

team. For the Decision Layer, granularity maps to the activities time-line being created and executed. Due to the nature of the dynamics of the physical system controlled by the Functional Layer, there is a strong correlation between its system granularity and the time-line granularity of the Decision Layer.

The blending of declarative and procedural techniques in the Decision Layer emerges from the trend of Planning and Scheduling systems that have Executive qualities and vice versa [30][14]. This has been afforded by algorithmic and system advances, as well as faster processing. CLARAty enhances this trend by explicitly providing for access of the Functional Layer at higher levels of granularity, thus less frequently, allowing more time for iterative replanning. However, it is still recognized that there is a need for procedural system capabilities in both the Executive interface to the Functional Layer, as well as the infusion of procedural semantics for plan specification and scheduling operations. Therefore, CLARAty has a single database to interface Planning and Executive Functionality, leveraging recent efforts to merge these capabilities [16].

The following sections will develop these concepts by providing an overview of features of both the Functional and Decision Layers, as well as the connectivity between them.

The Functional Layer

The Functional Layer is both an interface between the software and the hardware, and an interface for the Decision Layer to access basic capabilities of the system. Figure 3 shows a very simplified and stylistic representation of the Functional Layer. The Functional Layer has the following characteristics:

Object Oriented— Object oriented software design is desirable for several reasons. First, it can be structured to directly match the nested modularity of the hardware in a robotic system. Second, at all levels of this nesting, basic functionality and state information of the system components can be encoded and compartmentalized in its logical place. Third,

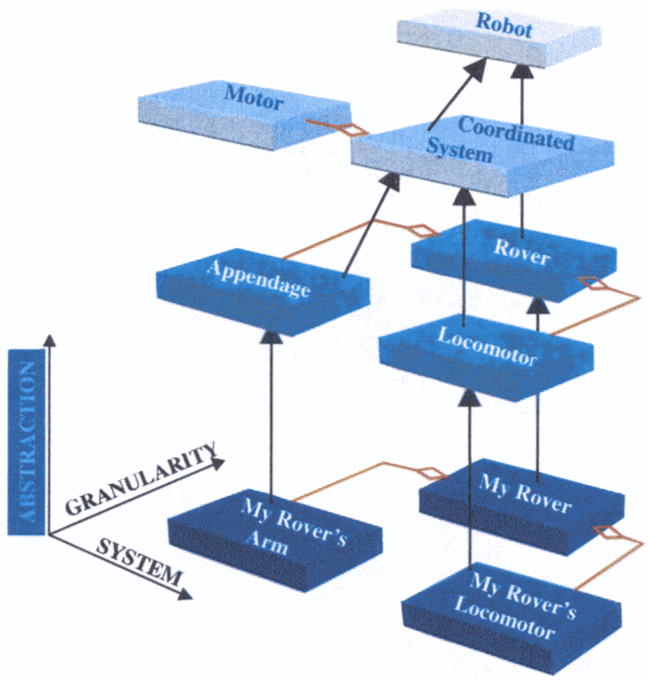

Figure 4. Simple example illustrating object hierarchy and Class inheritance concepts.

proper structuring of the software can use inheritance properties to manage the complexity of the software development. Finally, this structure can be graphically designed and documented using the UML standard.

Figure 4 gives a simplified description of the Object Hierarchy found in the Functional Layer. In this diagram, a fourth *Abstraction* Dimension has been added to illustrate the inheritance structure of the classes in the Functional Layer. At the bottom, a rover object aggregates arm and locomotor objects. While these objects comprise a specific *My Rover* system, each is derived from parent classes which are much more general.

An advantage of this structure is that it makes system extension much easier. First, multiple copies of the objects can be instantiated (e.g. two copies of *My Rover's Arm* — left and right). Second, two child classes may inherit all of the Appendage properties (e.g. *My Rover's Arm* and another class, *Your Rover's Arm*, where the latter is somewhat different from the former).

Moving up the class abstraction hierarchy, inheritance relationships may get more complicated. Both Appendage and Locomotor can have a common parent of Coordinated System, which in turn has the same parent as Rover, called Robot. Also, while the Motor class has no children, it is aggregated into the Coordinated System class. In this way, motor functionality is specified centrally in one object and available at all levels below it in the hierarchy, greatly simplifying software maintenance.

Encoded Functionality — All objects contain basic functionality for themselves, accessible from within the Functional Layer, as well as the Decision Layer. This functionality expresses the intended and accessible system capabilities. The purpose of this structure is to hide details from the higher levels of abstraction, as well as unifying the system structure. The latter is true when one member function name is used in all levels of the hierarchy, representing a capability that is appropriate for that level. Examples include: read, set, move, status, and so on.

For instance, a move command of a rover platform my be executed with a series of turns in place and straight line moves. This capability provides a basic service, while hiding the details of individual steering or wheel moves. However, it may not satisfy some special need of the Decision Layer, such as an arc trajectory. In this case, the option exists to access the subordinate wheel objects directly, to obtain the desired results.

Resident State — The state of the system components is contained in the appropriate object and obtained from it by query. This includes state variable values, state machine status, resource usage, health monitoring, etc. In this way, the Decision Layer can obtain estimates of current state or predictions of future state, for use in execution monitoring and planning.

Local Planners — Whereas the Decision Layer has a global planner for optimal decision making, it may utilize local planners that are part of Functional Layer Subsystems. For instance, path planners and trajectory planners, can be attached to manipulator and vehicle objects to provide standard capabilities without regard to global optimality. Like all other Functional Layer Infrastructure, the use of such local planners is an option for the Decision Layer.

Resource Usage Predictors — Similar to local planners, resource usage prediction is localized to the objects using the resources. Queries for these predictions are done by the Decision Layer during planning and scheduling, and can be requested at varying levels of fidelity. For instance, the power consumption by the vehicle for a particular traverse can be based on a hard-coded value, an estimate based on previous power usage, or a detailed analysis of the upcoming terrain.

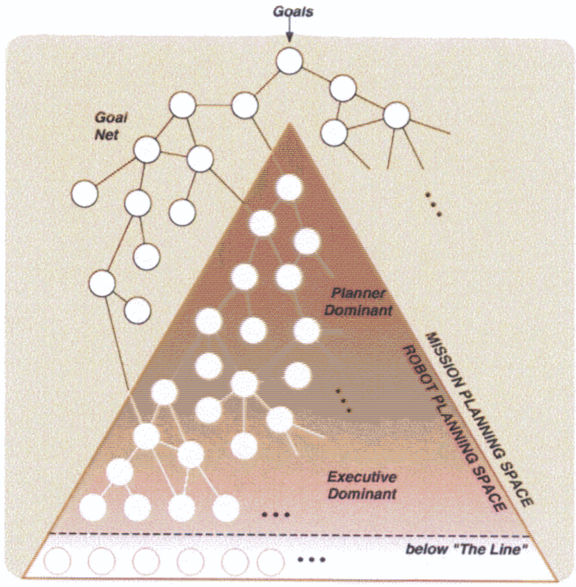

Figure 5. Proposed Decision Layer.

The level of fidelity requested will be based on time and resource constraints on the planning stage itself, margins available for the time window under consideration, as well as the availability of more detailed estimate infrastructure. In some cases, subordinate objects may be accessed by superior ones in the process of servicing a detailed prediction.

Simulation — In the simplest form, simulation of the system can be accomplished by providing emulation capability to all the lowest level objects that interact with hardware. In this case, the superior objects have no knowledge of whether they are actually causing real actions from the robot. Such simulation is a baseline capability of the architecture. However, it typically can not be done faster than real-time while using the same level of computer resources. Therefore, it is advantageous to percolate simulation capability up to superior objects in the hierarchy. The cost of this is increasing complexity in the simulation computations. For some purposes such complexity may be valuable. But, as with Resource Estimation, levels of fidelity may be specified to provide useful simulation with reduced computation when desired.

Test and Debug — For initial development and regression testing as system complexity grows, all objects must contain test and debug interfaces and have external exercisers.

The Decision Layer

The Decision Layer breaks down high level goals into smaller objectives, arranges them in time due to known constraints and system state, and accesses the appropriate capabilities of the Functional Layer to achieve them. Figure 5 shows a very simplified and stylistic representation of the Decision Layer. The Decision Layer has the following characteristics:

Goal Net — The Goal net is the conceptual decomposition of higher level objectives into their constituent parts, within the Decision Layer. It contains the declarative representation of the objectives during planning, the temporal constraint net-work resulting from scheduling, and possibly a task tree procedural decomposition used during execution.

Goals — Goals are specified as constraints on state over time. As such they can be thought of as bounding the system and specifying *what shouldn't be done*. They may be decomposed into subgoals during elaboration, and arranged in chronological order during scheduling. Resulting goal nets and schedules may be saved, or recalled [13].

Tasks — Tasks are explicitly parallel or sequential activities that are tightly linked. They result from the fixed procedural decomposition of an objective into a sequence, which is possibly conditional in nature. In contrast to Goals, Tasks specify *exactly what should be done* [30].

Commands — Commands are unidirectional specifications of system activity. Typically they provide the interface between the terminating fringes of the goal net, and the capabilities of the Functional Layer. Closed loop control within the Decision Layer is maintained by monitoring status and state of the system as commands are executed [4].

The Line — 'The Line' is the border between Decision-making and Functional execution [13]. It exists at the instantaneous lower border of the elaborated goal net, and moves to different levels of granularity according to the current elaboration. When projected on the Functional Layer, it denotes the border below which the system is a *black box* to the Decision Layer.

State — The state of the Functional Layer is obtained by query. The state of the Decision Layer, which is essentially its plan, the active elaboration, and history of execution, is maintained by this layer. It may be saved, or reloaded, in whole or part.

Layer Connectivity

Given the two architectural layers, Functional and Decision, there is flexibility in the ways in which these may be connected. At one end of the spectrum is a system with a very capable Decision Layer, and with a Functional Layer that provides only basic services. At the other end of the spectrum is a system with a very limited Decision Layer that relies on a very capable Functional Layer to execute robustly given high level commands. If both a capable Decision and Functional Layer are created then there may be redundancy — however, this is seen as a strength of CLARAty, not a weakness. It allows the system user, or the system itself, to consider the trade-offs in operating with the interface between the layers at a lower or higher level of granularity.

At lower granularity the built in capabilities of the Functional Layer are largely bypassed. This can enable the system to take advantage of globally optimized activity sequencing by the Decision Layer. It also enables the combination of latent functionality in ways that are not provided by aggregation of objects at higher levels of granularity in the Functional Layer. However, it requires that the Decision Layer be aware of all the small details of the system at lower granularity, and have

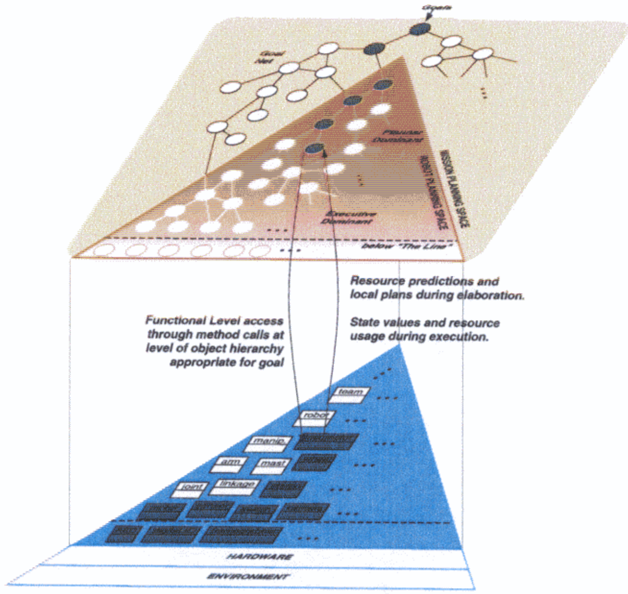

Figure 6. Proposed relationship of Function and Decision Layers.

time to process this information. For mission critical operations, it may be worth expending long periods of time to plan ahead for very short sequences of activity. However, this model can not be employed always, since it will force the system to spend a disproportionate amount of time planning, rather than enacting the plans. While the plan may provide optimality during its execution, inclusion of planning time as a cost may force the system be very suboptimal.

To avoid this problem of overburdening the Decision Layer, robust basic capabilities are built in to the Functional Layer for each object in its hierarchy. This allows the interface between the layers to exist at higher granularity. In this case, the Decision Layer need not second guess Functional Layer algorithms, and can also use more limited computing resources. Particularly in situations where resources usage is not near margins, or subsystems are not operating in parallel, it is much more efficient to directly employ the basic encoded functionality. It also directly allows for problem solving at the appropriate level of abstraction of the problem, both for the software and the developers.

Time-line Interaction

The interaction of the two architectural layers, can also be understood by considering the creation and execution of activities on a time-line. Figure 6 shows the two layers with the sequence of activation highlighted in green. In the Decision Layer, high level goals are decomposed into subordinate goals until there is some bottom level goal that directly accesses the Functional Layer. During planning and scheduling, this process occurs for queries of resource usage and local plans. If high fidelity information is requested from the Function Layer, such as when resource margins are tight, then the Functional Layer object may also need to access its subordinates to improve the predictions.

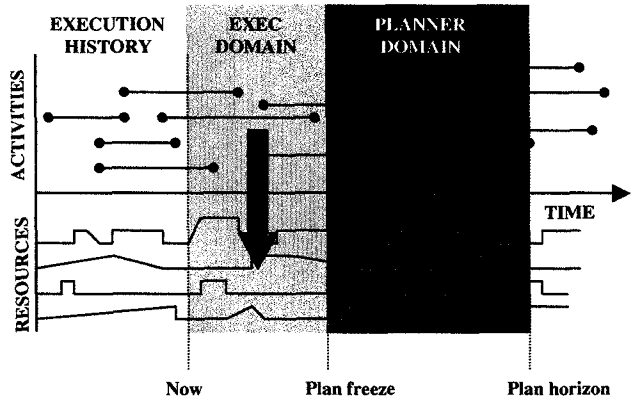

Figure 7. Example of system execution time-line.

The resultant activity list and resource usage is placed on a time-line as shown in Figure 7, activities on the top and resource usage on the bottom. Scheduling will optimally order these activities to enable goal achievement while not violating resource constraints. This process, however, must be frozen at some point sufficiently far in the future, so that the schedule is self-consistent at the time it is meant to be executed. Also, the time horizon up to which the planning and scheduling is done is also limited to constrain the problem. Both of these time boundaries are shown in the figure.

Inside the Plan Freeze boundary, it is the responsibility of an executive to initiate actions by accessing the Functional Layer. This process is illustrated in Figure 6 by the arrows to the Functional Layer, and the green shading of one portion of the object hierarchy it contains. As the actions take place, resources are consumed, typically in slightly different amounts than predicted. The usage is reported to the Decision Layer, where discrepancies are used to modify the future projections of resource availability on the time-line, forcing replanning to occur. This cycle is indicated by the green arrows in Figure 7

The process described is typical of systems where the procedural components of the executive are separated from the declarative components of planning and scheduling. It is not necessary that the boundary between planning and execution exist at a specific point in time — planning and scheduling can occur very near to the present, while executive-style procedural decomposition may be incorporated into future planning. Therefore, the plan freeze boundary in Figure 7 is not required for CLARAty, and the potential cross-coupling of Planner and Executive is one of the primary reasons for merging both into a single Decision Layer. The format of these merged activities, and the interface between them, is currently under development.

Finally, it is important to note that there is also a migration of some executive-style procedural expansion into the Functional Layer as well. Each object has built in functionality which will have a procedural decomposition of its actions, and may have it own mini-executive, or even planner. CLARAty does not preclude this, and allows for this functionality to be leveraged or bypassed, depending on the desire of system designers, and the capabilities of the Decision Layer.

4. Implementation

While the prototyping and implementation of the CLARAty architecture is still in its early stages, some specifications and results are important to mention, illustrating the direction of this work. Below are described some of the tool and standard choices, heritage software that will be included into the framework, and prototyping status at this time.

Tools and Standards

The following tools and standards have been accepted for CLARAty and its development:

The Universal Modeling Language — UML is to be used for system design and documentation. The intent is for full use of UML, including templates.

C++ Language — C++ will be used to create CLARAty, due to its wide use in academia and industry, the need for an object oriented implementation, and the requirements of real-time software implementation.

OS support — To provide both real-time software supports while allowing for workstation development, CLARAty will be constructed to run under VxWorks, Linux, and Solaris. Extension to other operating systems in the future is possible.

Standard Template Library — In the spirit of leveraging off public domain standards employed by the software community, software and specifications such as the Standard Template Library, will be employed where possible.

Software Development Tools — While it is possible to build all or parts of CLARAty by writing software directly with a text editor, it is desirable to employ a standard tool for organizing, structuring, and styling the software in a like manner across all developers. Consideration has been give to tools such as $Rhapsody^{TM}$ and $Visio^{TM}$, but no decision is final. Since it is the desire to not prevent wide participation in use of CLARAty, tools with large costs are not desirable.

Documentation — It is important to provide documentation of all components of the system in various forms. The UML was chosen partly for this reason. Other tools for in-line code documentation standardization are being investigated. The intent is to leverage current tools and standards, not to create new ones.

Heritage

While CLARAty is a new architecture design, its design and prototype construction will rely on some important existing infrastructure. First, some of the initial concepts for the Functional Layer object hierarchy were developed by the Planetary Dextrous Manipulators task at JPL [25]. Second, we will use the research rovers *Rocky 7* and *Rocky 8* to frame some of the problems, and as testbeds for prototyped solutions. Third, many years of technology development at JPL and other NASA research facilities have provided valuable software which will be implemented within the CLARAty framework. Among the software slated for inclusion is: JPL stereo vision [39] Carnegie Mellon University and JPL path planning [31][22], estimation [6], planning and scheduling [12], execution decomposition and monitoring [30], and kinematic and dynamics computing [41].

5. SUMMARY

This paper has presented our new CLARAty architecture for robotic autonomy software. We have briefly reviewed the history of this topic, potential impediments to success, needs for continued effort, and constraints on acceptable solutions. Given these circumstances, we have presented an evolutionary modification of prior architectural structure, which addresses the needs of merging procedural and declarative planning, while providing an object-oriented encapsulation of system functionality. The new CLARAty structure is, therefore, comprised of Decision and Functional Layers, and a complete overview of each of these, and their interaction, has been provided. Finally, a brief description of current implementation efforts was provided.

6. ACKNOWLEDGMENTS

As discussed, some of the concepts in this paper leverage those developed by the Mission Data System team at JPL. We would like to thank them for their continued interaction on architecture design issues.

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

REFERENCES

[1] R. Alami et al. An Archtecture for Autonomy. *International Journal of Robotics Research*, 17(4), April 1998.

[2] J. Albus, H. McCain, and R. Lumia. NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM). NBS Technical Note 1235, National Bureau of Standards, Gaithersburg, MD, July 1987.

[3] James Albus. 4-D/RCS Reference Model Architecture for Unmanned Ground Vehicles. In *IEEE Internation Conference on Robotics and Automation*, San Francisco, April 24-27 2000.

[4] P. Backes et al. Automated Planning and Scheduling for Planetary Rover Distributed Operations. In *IEEE Internation Conference on Robotics and Automation*, Detroit MI, May 1999.

[5] P. Backes, M. Long, and R. Steele. The Modular Telerobot Task Execution System for Space Telerobotics. In *IEEE Internation Conference on Robotics and Automation*, Atlanta Georgia, May 1993.

[6] J. Balaram. Kinematic State Estimation for a Mars Rover. *Robotica, Special Issue on Intelligent Autonomous Vehicles*, 18:251–262, 2000.

[7] J. Balaram and H. Stone. Automated Assembly in the JPL Telerobot Testbed. In *Intelligent Robotic Systems for Space Exploration*, Norwell, MA, 1992. Kluwer Academic Publishers.

[8] A. Bejczy. Robot Arm Dynamics and Control. Technical Memorandum 33-669, Jet Propulsion Laboratory, Pasadena, CA, February 1974.

[9] A. Bejczy and Z. Szakaly. An 8-D.O.F. Dual-Arm System for Advanced Teleoperation Performance Experiments. In *Fifth Annual Workshop on Space Operations Applications and Research (SOAR '91)*, Houston TX, July 9-11 1991.

[10] J. Borrelly et al. The ORCCAD Architecture. *International Journal of Robotics Research*, 17(4), April 1998.

[11] R. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal on Robotics and Automation*, 2(1), March 1986.

[12] S. Chien, R. Knight, R. Sherwood, and G. Rabideau. Integrated Planning and Execution for Autonomous Spacecraft. In *IEEE Aerospace Conference*, Aspen CO, March 1999.

[13] D. Dvorak, Rasmussen R., G. Reeves, and A. Sacks. Software Architecture Themes in JPL's Mission Data System. In *IEEE Aerospace Conference*, Big Sky, Montana, March 2000.

[14] T. Estlin, G. Rabideau, D. Mutz, and S. Chien. Using Continuous Planning Techniques to Coordinate Multiple Rovers. In *IJCAI Workshop on Scheduling and Planning*, Stockholm, Sweden, August 1999.

[15] R. Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, Department of Computer Science, 1989.

[16] F. Fisher et al. A Planning Approach to Monitor and Control for Deep Space Communications. In *IEEE Aerospace Conference*, Big Sky MT, March 2000.

[17] E. Gat. On Three-Layer Architectures. In D. Kortenkamp, R. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*, Boston, MA, 1998. MIT Press.

[18] E. Gat et al. Behavior Control for Robotic Exploration of Planetary Surfaces. *IEEE Transactions on Robotics and Automation*, 10(4):490–503, 1994.

[19] V. Hayward and R. Paul. Robot Manipulator Control Under Unix RCCL: A Robot Control "C" Library. *International Journal of Robotics Research*, 5(4), Winter 1986.

[20] http://www.intel.com/research/mrl/research/cvlib/. *Open Source Computer Vision Library*. Intel.

[21] R. Knight et al. Integrating Model-based Artificial Intelligence Planning with Procedural Elaboration for Onboard Spacecraft Autonomy. In *SpaceOps Conference*, Toulouse France, June 2000.

[22] S. Laubach. *Theory and Experiments in Autonomous Sensor-Based Motion Planning with Applications for Flight Planetary Microrovers*. PhD thesis, California Institute of Technology, May 1999.

[23] J. Matijevic et al. The Pathfinder Microrover. *Journal of Geophysical Research*, 102(E2):3989–4001, 1997.

[24] N. Muscettola. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence*, 103(1-2):5–48, 1998.

[25] I. Nesnas, M. Maimone, and H. Das. Rover Maneuvering for Autonomous Vision-Based Dexterous Manipulation. In *IEEE Conference on Robotics and Automation*, San Francisco, CA, April 2000.

[26] P. S. Schenker et al. FIDO Rover and Long-Range Autonomous Mars Science. In *Intelligent Robots and Computer Vision XVIII, SPIE Proceedings 3837*, Boston, September, 1999.

[27] S. Schneider et al. ControlShell: A Software Architecture for Complex Electromechanical Systems. *International Journal of Robotics Research*, 17(4), April 1998.

[28] M. Schoppers. A Software Architecture for Hard Real-Time Execution of Automatically Synthesized Plans or Control Laws. In *AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS)*, Houston TX, March 20-24 1994.

[29] R. Simmons. Structured Control for Autonomous Robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, 1994.

[30] R. Simmons and D. Apfelbaum. A Task Description Language for Robot Control. In *IEEE/RSJ Intelligent Robotics and Systems Conference*, Vancouver Canada, October 1998.

[31] S. Singh et al. Recent Progress in Local and Global Traversability for Planetary Rovers. In *IEEE Conference on Robotics and Automation*, San Francisco, CA, April 2000.

[32] E. Tunstel, R. Welch, and B. Wilcox. Embedded Control of a Miniature Science Rover for Planetary Exploration. In *7th International Symposium on Robotics with Applications, WAC'98*, Anchorage Alaska, May 1998.

[33] R. Volpe. Navigation Results from Desert Field Tests of the Rocky 7 Mars Rover Prototype. *International Journal of Robotics Research*, 18(7), 1999.

[34] R. Volpe and J. Balaram. Technology for Robotic Surface Inspection in Space. In *AIAA Conference on Intelligent Robots in Feild, Factory, Service, and Space (CIRFFSS)*, Houston, Texas, March 20-24 1994.

[35] R. Volpe et al. A Prototype Manipulation System for Mars Rover Science Operations. In *IEEE/RSJ International Conference on Robots and Systems (IROS)*, Grenoble, France, September 7-11 1997.

[36] R. Volpe et al. Rocky 7: A Next Generation Mars Rover Prototype. *Journal of Advanced Robotics*, 11(4):341–358, 1997.

[37] B. Wilcox et al. Robotic Vehicles for Planetary Exploration. In *IEEE Conference on Robotics and Automation*, pages 175–180, Nice France, May 12-14 1992.

[38] D. Woerner. X2000 Systems And Technologies For Missions To The Outer Planets. In *49th International Astronautical Congress/International Astronautica Federation*, Melbourne, Australia, September 28-October 2 1998.

[39] Y. Xiong and L. Matthies. Error Analysis of a Real-Time Stereo System. In *Computer Vision and Pattern Recognition*, pages 1087–1093, 1997.

[40] Y. Xiong and L. Matthies. Vision-guided Autonomous Stair Climbing. In *IEEE Conference on Robotics and Automation*, San Francisco, CA, April 2000.

[41] J. Yen and A. Jain. ROAMS: Rover Analysis Modeling and Simulation Software. In *nternational Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS'99)*, Noordwijk, Netherlands, June 1999.

BIOGRAPHIES

Richard Volpe, Ph.D., is the Principal Investigator for the Long Range Science Rover Research Team. His research interests include real-time sensor-based control, robot design, software architectures, path planning, and computer vision. Richard received his M.S. (1986) and Ph.D. (1990) in Applied Physics from Carnegie Mellon University, where he was a US Air Force Laboratory Graduate Fellow. His thesis research concentrated on real-time force and impact control of robotic manipulators. Since December 1990, he has been at the Jet Propulsion Laboratory, California Institute of Technology, where he is a Senior Member of the Technical Staff. Until 1994, he was a member of the Remote Surface Inspection Project, investigating sensor-based control technology for telerobotic inspection of the International Space Station. Starting in 1994, he led the development of Rocky 7 and 8, next generation mobile robot prototypes for extended-traverse sampling missions on Mars. In 1997, he received a NASA Exceptional Achievement Award for this work, which has led to the design concepts for the 2003 Mars rover mission.

Issa A.D. Nesnas, Ph.D., is the cognizant engineer for the Architecture and Autonomy Research task. His research interests include software and hardware architectures for robotic systems, autonomous sensor-based coordination, actuation and control, computer vision, object-oriented design, and artificial intelligence. Issa received a B.E. degree in Electrical Engineering from Manhattan College, NY, in 1991. He earned the M.S. and Ph.D. degrees in robotics from the Mechanical Engineering Department at the University of Notre Dame, IN, in 1993 and 1995 respectively. In 1995, he joined Adept Technology Inc. as a senior project engineer inventing and implementing several new technologies for high-speed vision-based robotic applications and holds a patent for the Impulse-based flexible parts feeder. He also participated in the NCMS Consortium working with industry leaders in factory automation such as Ford, GM, Delco Electronics, and Cummins Engine designing hardware and software standards for robotic assembly cells. He has joined NASA at the Jet Propulsion Laboratory as a member of technical staff in 1997. At JPL he has worked on the Planetary Dexterous Manipulator project researching autonomous sensor-based manipulation for rovers. In addition to his duties on the Architecture and Autonomy task, Issa is also working on a flight project for autonomous sensor-based landing on Mars. He has received several Notable Organizational Value Added (NOVA) Awards and an Exceptional Achievement Award for his work at JPL. Issa is a member of Eta Kappa Nu and Tau Beta Pi National Honor Societies.

Tara Estlin, Ph.D., is a senior member of the Artificial Intelligence Group at the Jet Propulsion Laboratory in Pasadena, California where she performs research on planning and scheduling systems for rover automation and multi-rover coordination. Dr. Estlin has led several JPL efforts on automated rover-command generation and planning for distributed rovers, and is a team member of the JPL Long Range Science Rover team. She received a B.S. in computer science in 1992 from Tulane University, an M.S. in computer science in 1994 and a Ph.D. in computer science in 1997, both from the University of Texas at Austin. She has numerous publications in planning/scheduling and machine learning, including such high profile forums such as AAAI, IJCAI, and ICRA. Her current research interests are in the areas of planning, scheduling, and multi-agent systems.

Darren Mutz received the Bachelor of Science degree in Computer Science at the University of California, Santa Barbara in 1997. Current projects include Long Range Science Rover (LRSR), statistical hypothesis evaluation, ASPEN.

Richard Petras is a member of the Telerobotics Research and Applications Group at the Jet Propulsion Laboratory, California Institute of Technology.

Hari Das, received his ScD degree on Mechanical Engineering from MIT in 1989. He is a Senior Member of Technical Staff at JPL. His research interests are in the development and evaluation of robotic systems for biomedical and planetary exploration applications.